SAGE :
a Sentence Parsing and Generation System

Jean-Marie Lancel, Miyo Otani, Nathalie Simonin
Cap Sogeti Innovation
118 rue de Tocqueville, 75017 Paris, France
E-mail: lancel@csinn.uucp, otani@csinn.uucp, simonin@csinn.uucp

Laurence Danlos
LADL – CNRS
Tour Centrale, Université Paris VII, 4 place Jussieu, 75005 Paris, France

Abstracts:

SAGE (Sentence Analysis and GEneration system) is an operational parsing and generating system. It is used as a Natural Language Frontend for Esprit project Esteam–316, whose purpose is to advise a novice user through a cooperative dialogue.

The aim of our system is to validate the use of a Lexicon–Grammar (drawn from the LADL studies) for sentence–parsing and generation, and to implement linguistic knowledge in a declarative way using a formalism based upon Functional Descriptions (FD). We have also developed the parser and the generation module so that they share informations and knowledge bases as much as possible: they work on the same semantic dictionary and the same linguistic knowledge bases, except that they have their own grammar. We have also implemented a tracking of semantic objects that have been instantiated during a dialogue session: the so–called Token History is provided for semantic reference and anaphor resolution during parsing and for pronoun production during generation.

After introducing to Esteam–316, this paper describes the linguistic knowledge bases required by SAGE, and then focuses on the Generation Module. Section 4 explains how pronouns are handled. The last section is a brief evaluation of our present work.

1 Introduction to the application of SAGE

The parsing and generating system described here is used as a Natural Language Frontend for Esprit project Esteam–316, which is an Advice–Giving system [Decitre 87] [Bruffaerts 86]. A cooperative interactive Man–Machine Interface carries out dialogue functionalities such as recognition of user queries, explanation of domain–concepts, explanation of solutions proposed by the Problem Solver, etc. To describe it briefly, this Dialogue Manager handles the pragmatic level of a dialogue, whereas the Natural Language Frontend SAGE deals with linguistic inferences. The chosen language is English.

The Dialogue Manager and SAGE share the same semantic objects, using a formalism based upon Functional Descriptions (FDs) [Kay 81]. The Parser of SAGE extracts the meaning of the user's query and represents it with nested FDs. On the other hand, the Dialogue Manager sends the Generator FDs which describe the semantic contents of the answer.

Our previous work [Lancel 86] was based on a unique dictionary and a grammar shared by both a parser and a generation module. The grammar formalism required the mixing of syntactic and semantic informations in the same structure, which implied the complete rewriting of grammar when changing from one application domain to another. It could not handle transformational processes such as interrogative and imperative transformations. The system presented here fulfills the four following requirements:

1. definition of linguistic knowledge bases suitable for both parsing and generation;
2. integration of lexicon–grammar theory into the previous formalism, in order to provide precise syntactic information;
3. modularization: a change of application

should not lead to a complete rewriting, but only to an extension of the semantic levels;

4. proper pronoun handling, both when parsing (reference resolution) and when generating (pronoun synthesis).

The section 2 describes the linguistic dictionaries of SAGE. The section 3 explains how those dictionaries are exploited by the generation module. In the section 4, we will detail what kind of processes are required by pronoun handling.

2 Linguistic knowledge base for parsing and generation

There are three linguistic levels handled by our system: morphological, syntactic, and lastly semantic. The first one will not be explained here, since the most innovative aspects of our linguistic knowledge bases are provided by the two other levels: we are able to take into account a wide range of constructions of a given language using the lexicon–grammar and we use a totally declarative formalism.

2.1 Parsing versus generation

The main feature of SAGE is that the Parsing and Generation processes are carried out using the same dictionaries.

These dictionaries are interpreted by two separate grammars, one for parsing and one for generation, both of them being language–dependent but not domain–dependent. This is a major conclusion drawn from our studies: a parser and a generation module can hardly share the same grammar rules, for the heuristics required by these two processes are fundamentally different. Unlike parsing, a generation process has nothing to do with a sequence of "left–to–right" procedures [Danlos 87a, Danlos 87b]. Moreover, a given heuristic of clause transformation is strongly dedicated to a parsing or to a generation process (see section 3).

2.2 Syntactic Knowledge Base

This syntactic level is domain–independent: constructions of verbs, predicative nouns and adjectives along with their corresponding valency are listed in a *lexicon–grammar*.

This lexicon–grammar is based on the theory developed by [Gross 75, Gross 86] and the studies carried out by the LADL on French constructions. It provides accurate specifications of the acceptable valencies, with different levels of correctness for parsing and generation. This allows a very wide range of sentence structures in generation, and semantic inferences to avoid ambiguities. The LADL Lexicon–Grammar covers nearly all French constructions. As far as we know, an equivalent amount of work is still not available for English. Therefore, we developed a Lexicon–Grammar containing a few English verbs and nouns. The corresponding constructions are drawn from [LONGMAN 81].

To give an idea of this lexicon–grammar, we present below the information stored for the verb *want*.

- The standard construction is [Subject + Verb + Direct Object];
- The subject must be a human being; therefore, it is allowed to be a noun group, but not a clause or a verb phrase;
- The direct object may be a human being as in "The mother wants *her child*", a non–human entity as in "He wants *time*", or a *that–clause* as in "Mary wants *that John settles down in Paris*";
- The *that–clause* can be reduced in the following forms:
  - [Noun group + Adjective] or *NAdj* if the verb is *be* (e.g "The teacher wants *the exercise ready for tomorrow*");
  - [Verb at the complete infinitive form + complements] or *ToVinf0* if the concept of the subject of this clause is the same as that of the subject of *want* (e.g. "Mary wants *to settle down in Paris.*");
  - [Noun group + Verb at the complete infinitive form + complements] or *NToVinf* when the two subjects are different (e.g. "Mary wants *her friends to settle down in Paris.*");
- The whole clause may be transformed into the passive form.

For the sake of readability and maintenance, verbs are sorted into different *tables*. One table specifies several standard features, syntactic constructions as well as valencies that are common to every verb of the same table. For instance, in our lexicon–grammar, *want* belongs to the table *table_NVS* whose standard construction is a human subject in a noun group, with a non–human direct object in a noun group, construction of which may be transformed into the passive form.

Here is how one construction of the verb *want* is coded, using *Functional Descriptions* (FD):

```
syntax_want ←→
  [ table = tab_NVS
    linguistic_definition = clause_want
    verb = [ word = want ]
    object1 =
      [ reduction = {(ToVinf0 100) (NAdj 100)}
        clause = {(NToVinf 100)} ] ]

tab_NVS ←→
```

```
[ subject =
    [ distribution = {(*human 100)}
      noun_phrase = {(noun_phrase 100)} ]
  object1 =
    [ distribution = { (*non_human 100)
                       (*human 100) }
      noun_phrase = {(noun_phrase 100)} ]
  transformation = {(passive 100)} ]
```

The lexical codes (*NToVinf, ToVinf0* and *NAdj*) are specified in a FD, stating the conditions of validity of the code and the consequences on the components: *ToVinf0* should be chosen if the subject of the current sentence and that of the main clause represent the same concept; in this case the subject should be omitted and the verb should be in the infinitive form.

The numeric values ranging from 0 up to 100, is a coefficient on the correctness of the corresponding constructions. When generating a syntactic component, lexical codes that are allowed are the ones with a coefficient greater than a certain value, 70 in our implementation. When parsing sentences, accepted constructions would be of a coefficient greater than another milestone, 30 for instance. The values 0, 30, 70, 100 are of course quite arbitrary. But they allow the parsing of constructions that are often understood by most of the people but are syntactically incorrect: the corresponding lexical codes will have a coefficient between 30 and 70.

2.3 Semantic knowledge base

The semantic level is highly domain–dependent since it deals with concepts. The application domain chosen by the Esteam–316 project is financial advice–giving for non–expert users. Therefore, the Man–Machine interface handles intention concepts such as **surface_request* and **surface_inform* which are the *intention of asking for something* and the *intention of stating something*, financial concepts such as **emergency_fund* which is a *certain amount of money available at any time and provided for emergency cases*, and lastly domain–independent concepts such as **want*[1].

Those concepts are organised in a semantic network, using the links *is_a* and *example*. Moreover, the semantic sub–items are specified in a *schema*. For instance, the concept **want* is specified by:

```
*want <-->
  [ is_a = *position
    schema =
      [ actor = [ is_a = *human ]
        object                    ]
    synthesis =
      ( [ linguistic_definition = clause_want ] ) ]
```

[1] The chosen convention is to put a star at the beginning of a concept identifier, but this is purely for the sake of readability.

As seen in section 2.1, the semantic objects actually handled by the user and system during a dialogue are called *tokens*. Inside the system, tokens are instances of concepts — or more precisely of schemata —.

2.4 Link between concepts and syntactic structures

Mapping between semantic schemata and syntactic structures is specified in FDs named *linguistic definitions*. This is an important feature of our KB: it is the linguistic definitions that make explicit the correspondance between token slots and syntactic components of sentences, clauses and noun groups. Using them, the same token may be synthesised as a noun phrase or a clause, according to syntactic constraints. A noun phrase or a clause require different grammar rules in the generation process.

For instance, let us consider the following token :

```
[ instance_of = *transaction
  buyer = *Mary
  object = *car ]
```

It may be expressed with a clause *Mary buys a car*:

```
[ meaning =
    [ instance_of = *transaction
      buyer = *Mary
      object = *car ]
  subject = [ meaning = *Mary ]
  verb = [ word = buy ]
  object = [ meaning = *car ] ]
```

or with a noun phrase *Mary's purchase of a car*:

```
[ meaning =
    [ instance_of = *transaction
      buyer = *Mary
      object = *car ]
  subjective_genitive = [ meaning = *Mary ]
  predicative_noun = [ word = purchase ]
  objective_complement = [ meaning = *car ] ]
```

The last two FDs shown above are the syntactic structures produced by two different linguistic definitions linked to the same concept **transaction*. The choice between the two is made by the generation module either under semantic constraints declared in the semantic dictionary, or under linguistic restrictions specified by the generation grammar, or by the lexicon–grammar.

Linguistic definitions do not only allow the synthesis of totally different schemata using the same generation grammar rules, but also provide the parser with extended capacities for handling complex noun phrases or sentences and for extracting a specific meaning with the specific slot identifiers (*buyer, object, year*) out of a standard syntactical construction of the noun– or verb–predicate.

3 Generation

3.1 General heuristic of the Generation Module

The generation process is top–down, with backtracking. The generation algorithm consists of building a complex object of several nested FDs recursively.

The highest level deals with the surface syntactic form: assertion, question, order. This level corresponds to the intention concepts like **surface_request*. Then comes the inner structure of the sentence: generally speaking, a subject, a verb and objects with several optional adverbials. This corresponds to domain–concepts (e.g. **emergency_fund*) or general concepts (e.g. **want*). Lastly there is the noun group structure with preposition, determiner, noun. There is a specific grammar rule for each level.

Briefly, a grammar rule specifies under what conditions a given rule may be applied, what kinds of rules are to be chosen for the synthesis of each Syntactic Component, and what actions are to be carried out on the structure (such as choosing the number and person of a verb according to the subject within a sentence).

The current level is built in a loop starting from its semantic contents (a token): through the concept corresponding to the token, the interpreter chooses a linguistic definition, then a syntactic structure in the lexicon–grammar.

These FDs plus the corresponding grammar rule are functionally unified[2] with the current object. Then, one syntactic code such as *ToVinf0* or *noun_phrase* is chosen according to the grammar rule and the validity condition of the code. The FD of this code is unified with the current object.

This is where our declarative KBs based on Functional Descriptions prove to be efficient. The same heuristic based on functional unification is used for totally different structures such as noun phrase or clause. Therefore, this loop is allowed to be totally recursive.

[2] in the meaning of *functional unification* [Kay 81].

At this stage of the process, the generation module may add several modifiers to the current level, that are adverbials in sentences, or adjectives in noun groups: this adjunction is also carried through functional unification since the modifiers are also described in a FD just like any grammar rule or lexical code.

For instance after functional unifications, the current syntactic component corresponding to *"I want a car"* is:

```
[ meaning =
    [ instance_of = *want
      actor = *user
      object = [ instance_of = *car ] ]
  subject =
    [ meaning = *user
      distribution = {(*human 100)}
      noun_phrase = {(noun_phrase 100)} ]
  verb = [ word = want ]
  object1 =
    [ meaning = [ instance_of = *car ]
      distribution = { (*non_human 100)
                       (*human 100)}
      reduction = { (ToVinf0 100)
                    (NAdj 100)}
      clause = {(NToVinf 100)}
      noun_phrase = {(noun_phrase 100)} ]
  transformation = {(passive1 100)} ]
```

Then transformations are processed whenever they are needed, such as for questions (which puts the verb in the interrogative form and inserts an auxiliary verb before the subject), or negations or passive transformations. Transformations are specified in FDs similar to grammar rules, with validity conditions and actions, but also with a specific slot stating whether they must be applied before or after the standard grammar rules.

This synthesis loop is carried out on every syntactic sub–component, that is for instance on subject, verb and objects of a clause.

If every sub–component is correctly synthesised in turn, the actions of the global rule are applied on the current component.

Other transformations may be carried out, leading only to the re-ordering of objects in a clause, which may depend on whether the objects are expressed through pronouns. A ditransitive/dative transformation is a perfect example: starting from a sentence whose meaning is *"The postman gives Mary the letter"*, the final sentence may become *"The postman gives her the letter"* or *"The postman gives it to Mary"* or *"The postman gives it to her"*.

There ends the body of the loop. If a failure occurs during this loop, backtracking chooses another linguistic definition and/or another grammar rule.

3.2 One example of input of a sentence generation

The following token, made of several nested tokens, is synthesised as "*Under what delay do you want your emergency fund available?*" is:

```
[ instance_of = *s_request
  speaker = *system
  hearer = *user
  force = weak
  proposition =
    [ instance_of = *inform_ref
      actor = *user
      proposition =
        [ instance_of = *want
          actor = *user
          proposition =
            [ instance_of = *emergency_fund
              agent = *user
              delay = *unknown ] ] ] ]
```

The combination of the tokens **surface_request* and **inform_ref* produces a Wh-question. The question focuses on the *delay* of the token **emergency_fund* as indicated by the special object **unknown*: this transforms the adverbial of delay of the structure "the emergency fund is available *in dd days*" into the compound interrogative pronoun *under what delay*. The interrogative pronoun is moved to the beginning of the sentence, coming from the nested infinitive clause *under what delay is your emergency fund available?*. As the verb of the clause expressing the token **emergency_fund* is *to be*, the construction adopted for the direct object of the verb *want* is *NAdj*: the verb *to be* is removed. The possessive *your* is synthesised from the slot *agent* of the token **emergency_fund*.

4 Pronoun handling in parsing and generation

4.1 Token History

Pronoun handling requires the recording of all the concept instances (tokens). A token may be an entity (e.g. an instance of the concept **car*) or a relation between entities (e.g. an instance of the concept **want*). In the sentence *I know that you made a wrong investment.*, the tokens are the *system* ("I"), the *user* ("you"), the *investment* ("a wrong investment"), and also the relation introduced by *that* ("you made a wrong investment"), and the relation corresponding to the whole sentence.

During a dialogue, the system records these tokens in a *Token History*. Besides the tokens themselves, other linguistic information is needed for reference resolution. The characteristics stored for each token are the turn number within the dialogue (a turn is over whenever one of the two locutors has finished speaking), the sentence number within the turn, the locutor (during parsing, the locutor is the user, whereas during generation, it is the system), the type of the token (entity or relation), and the linguistic expression (noun phrase, pronoun, demonstrative pronoun, clause).

The Token History is updated by three processes: the parsing module, the application (here the Esteam-316 Dialogue Manager) and the generation module. Of course, it is very important for the Dialogue Manager that if one token is produced by the parsing of one sentence, then the generation module would synthesise the same sentence from the same token.

After analysis of the user's sentence, the History is updated with the tokens of the sentence, which are all first considered as new. The Dialogue Manager receives the new tokens, sometimes with a list of former tokens to which a given new one may refer: a typical case is when a pronoun is found in the user's sentence; then the parser has to resolve references on morphological, syntactic and semantic grounds, in order to prepare the dialoger's pragmatic inference. It is the Dialogue Manager which is in charge of defining the final status of each new token through pragmatic inferences:

- when it corresponds to a pronoun, the token to which it refers
- otherwise, whether it is a redefinition of a token previously used, or a totally new one.

If a sentence generation succeeds, the generation module updates the History with the linguistic information of the synthesised tokens.

4.2 Pronoun synthesis

The generation grammar checks whether each item to be generated may be synthesised by a pronoun.

The first step is to choose the appropriate pronoun. The second step consists of verifying that the chosen pronoun will not be ambiguous for the user according to the History of Tokens.

The computing of the morphological form of the pronoun and the checking of ambiguity are very complex and require the handling of semantic, syntactic and morphological constraints. For precise explanations and comparison with other studies, see [Danlos 88].

5 Evaluation of SAGE and its generation Module

The parsing and generation grammar formalism are intended to support a changing from English to French. For instance, both the order of synthesis of the syntactic components of a clause or a noun phrase and the pronoun synthesis control are specified declaratively. This allows the reusability and adaptability of this Natural Language Frontend through the creation of an adapted semantic dictionary and the extension of grammars, provided that the application is able to make inferences on semantic, or even pragmatical levels (which is the case of Esteam-316 Dialogue Manager).

SAGE runs on Sun workstations. It is able to parse complex assertions (*I want to buy a car in five years.*), Yes/No questions (*Could I put 500 dollars into my emergency-fund?*), and acknowlegement expressions (*Yes. No. OK!*).

It can synthesise complex assertions with infinitive clauses and adverbials, imperative sentences, Yes/No-questions, and Wh-questions. The interrogative pronouns of Wh-questions may stem either from the main clause (as in *What do you buy?*) or from nested clauses (as in *How much do you want to invest?*). As far as we know in the generation realm, it seems that the most similar work is the synthesis system PHRED citejacobs. Sentence production in PHRED is a recursive process divided into three phases: 1) pattern-concept fetching, 2) pattern restriction, and 3) pattern interpretation. Their objectives are similar to 1) the choice of a linguistic definition, 2) the verification of semantic distribution and the application of a lexical code on the Syntactic Component, 3) the generation of the syntact sub-components. Other studies (Danlos, McKeown, Appelt) are more related to the strategies for text production than to sentence generation heuristics.

It can also synthesise complex assertions with infinitive clauses and adverbials, imperative sentences, Yes/No-questions, and Wh-questions. The interrogative pronouns of Wh-questions may stem either from the main clause (as in *What do you buy?*) or from nested clauses (as in *How much do you want to invest?*).

Pronoun handling is currently developed.

References

[Bruffaerts 86] Bruffaerts A., Henin E. and Marlair V., *An Expert System Prototype for Financial Counseling*, Research Report 507, Philips Research Laboratory Brussels, 1986.

[Danlos 88] Laurence Danlos, Fiammetta Namer, *Morphological and cross dependencies in the synthesis of personal pronouns in Romance languages*, Coling'88.

[Danlos 87a] Laurence Danlos, *A French and English Syntactic Component for Generation*, Natural Language Generation: New Results in Artificial Intelligence, Psychology and Linguistics, Kempen G. ed, Dortrecht/Boston, Martinus Nijhoff Publishers, 1987.

[Danlos 87b] Laurence Danlos, *The linguistic basis of text generation*, Cambridge University Press.

[Decitre 87] Paul Decitre, Thomas Grossi, Cléo Jullien, Jean-Philippe Solvay, *Planning for Problem-Solving in Advice-Giving Dialogue*, ACL European Chapter, Copenhague, 1987.

[Gross 86] Maurice Gross, *Lexicon-Grammar, The Representation of Compound Words*, 11th International Conference on Computational Linguistics, Proceedings, Coling'86, 1986.

[Gross 75] Maurice Gross, *Méthodes en syntaxe, Régime des constructions complétives*, Hermann, 1975.

[Jacobs 85] Paul S. Jacobs, *PHRED: A Generator for Natural Language Interfaces*, Computational Linguistics, Vol. 11, No 4, 1985.

[Kay 81] Martin KAY, *Unification Grammars*, Xerox Publication, 1981.

[Lancel 86] Jean-Marie Lancel, Francois Rousselot, Nathalie Simonin, *A Grammar used for Parsing and Generation*, 11th International Conference on Computational Linguistics, Proceedings, Coling'86, 1986.

[Longman 81] *Longman Dictionary of Contemporary English*, Longman Group Limited, 1978, Corrections 1981.

[Simonin 87] Nathalie Simonin, *An Approach of Creating Structured Text*, First European Workshop on Natural Language Generation, Royaumont Abbey, 1987.